mdn web docs

Protocol upgrade mechanism

The HTTP/1.1 protocol provides a special mechanism that can be used to upgrade an already established connection to a different protocol, using the `Upgrade` header field.

This mechanism is optional; it cannot be used to insist on a protocol change. Implementations can choose not to take advantage of an upgrade even if they support the new protocol, and in practice, this mechanism is used mostly to bootstrap a WebSockets connection.

Note also that HTTP/2 explicitly disallows the use of this mechanism; it is specific to HTTP/1.1.

Upgrading HTTP/1.1 Connections

The `Upgrade` header field is used by clients to invite the server to switch to one of the listed protocols, in descending preference order.

Because `Upgrade` is a hop-by-hop header, it also needs to be listed in the `Connection` header field. This means that a typical request that includes Upgrade would look something like:

HTTP
```
GET /index.html HTTP/1.1
Host: www.example.com
Connection: upgrade
Upgrade: example/1, foo/2
```

Other headers may be required depending on the requested protocol; for example, WebSocket upgrades allow additional headers to configure details about the WebSocket connection as well as to offer a degree of security in opening the connection. See Upgrading to a WebSocket connection for more details.

If the server decides to upgrade the connection, it sends back a `101 Switching Protocols` response status with an Upgrade header that specifies the protocol(s) being switched to. If it does not (or cannot) upgrade the connection, it ignores the `Upgrade` header and sends back a regular response (for example, a `200 OK`).

Right after sending the `101` status code, the server can begin speaking the new protocol, performing any additional protocol-specific handshakes as necessary. Effectively, the connection becomes a two-way pipe as soon as the upgraded response is complete, and the request that initiated the upgrade can be completed over the new protocol.

Common uses for this mechanism

Here we look at the most common use cases for the `Upgrade` header.

Upgrading to a WebSocket connection

By far, the most common use case for upgrading an HTTP connection is to use WebSockets, which are always implemented by upgrading an HTTP or HTTPS connection. Keep in mind that if you're opening a new connection using the WebSocket API, or any library that does WebSockets, most or all of this is done for you. For example, opening a WebSocket connection is a single method:

JS
```
webSocket = new WebSocket("ws://destination.server.ext", "optionalProtocol");
```

The `WebSocket()` constructor does all the work of creating an initial HTTP/1.1 connection then handling the handshaking and upgrade process for you.

> **Note:** You can also use the `"wss://"` URL scheme to open a secure WebSocket connection.

If you need to create a WebSocket connection from scratch, you'll have to handle the handshaking process yourself. After creating the initial HTTP/1.1 session, you need to request the upgrade by adding to a standard request the `Upgrade` and `Connection` headers, as follows:

HTTP
```
Connection: Upgrade
Upgrade: websocket
```

WebSocket-specific headers

The following headers are involved in the WebSocket upgrade process. Other than the `Upgrade` and `Connection` headers, the rest are generally optional or handled for you by the browser and server when they're talking to each other.

`Sec-WebSocket-Extensions`

Specifies one or more protocol-level WebSocket extensions to ask the server to use. Using more than one `Sec-WebSocket-Extension` header in a request is permitted; the result is the same as if you included all of the listed extensions in one such header.

HTTP
```
Sec-WebSocket-Extensions: extensions
```

`extensions`

A comma-separated list of extensions to request (or agree to support). These should be selected from the IANA WebSocket Extension Name Registry . Extensions which take parameters do so by using semicolon delineation.

For example:

HTTP
```
Sec-WebSocket-Extensions: superspeed, colormode; depth=16
```

`Sec-WebSocket-Key`

Provides information to the server which is needed in order to confirm that the client is entitled to request an upgrade to WebSocket. This header can be used when insecure (HTTP) clients wish to upgrade, in order to offer some degree of protection against abuse. The value of the key is computed using an algorithm defined in the WebSocket specification, so this *does not provide security*. Instead, it helps to prevent non-WebSocket clients from inadvertently, or through misuse, requesting a WebSocket connection. In essence, then, this key confirms that "Yes, I really mean to open a WebSocket connection."

This header is automatically added by clients that choose to use it; it cannot be added using the `fetch()` or `XMLHttpRequest.setRequestHeader()` methods.

HTTP
```
Sec-WebSocket-Key: key
```

`key`

The key for this request to upgrade. The client adds this if it wishes to do so, and the server will include in the response a key of its own, which the client will validate before delivering the upgrade response to you.

The server's response's `Sec-WebSocket-Accept` header will have a value computed based upon the specified `key`.

`Sec-WebSocket-Protocol`

The `Sec-WebSocket-Protocol` header specifies one or more WebSocket protocols that you wish to use, in order of preference. The first one that is supported by the server will be selected and returned by the server in a `Sec-WebSocket-Protocol` header included in the response. You can use this more than once in the header, as well; the result is the same as if you used a comma-delineated list of subprotocol identifiers in a single header.

HTTP

```
Sec-WebSocket-Protocol: subprotocols
```

`subprotocols`

A comma-separated list of subprotocol names, in the order of preference. The subprotocols may be selected from the IANA WebSocket Subprotocol Name Registry or may be [illegible] jointly understood by the client and the server.

`Sec-WebSocket-Version`

REQUEST HEADER

Specifies the WebSocket protocol version the client wishes to use, so the server can confirm whether or not that version is supported on its end.

HTTP

```
Sec-WebSocket-Version: version
```

`version`

The WebSocket protocol version the client wishes to use when communicating with the server. This number should be the most recent version possible listed in the IANA WebSocket Version Number Registry . The most recent final version of the WebSocket protocol is version 13.

RESPONSE HEADER

If the server can't communicate using the specified version of the WebSocket protocol, it will respond with an error (such as 426 Upgrade Required) that includes in its headers a `Sec-WebSocket-Version` header with a comma-separated list of the supported protocol versions. If the server does support the requested protocol version, no `Sec-WebSocket-Version` header is included in the response.

HTTP

```
Sec-WebSocket-Version: supportedVersions
```

`supportedVersions`

A comma-delineated list of the WebSocket protocol versions supported by the server.

Response-only headers

The response from the server may include these.

`Sec-WebSocket-Accept`

Included in the response message from the server during the opening handshake process when the server is willing to initiate a WebSocket connection. It will appear no more than once in the response headers.

HTTP

```
Sec-WebSocket-Accept: hash
```

`hash`

If a `Sec-WebSocket-Key` header was provided, the value of this header is computed by taking the value of the key, concatenating the string "258EAFA5-E914-47DA-95CA-C5AB0DC85B11" to it, taking the SHA-1 hash of that concatenated string, resulting in a 20-byte value. That value is then base64 encoded to obtain the value of this property.

Specifications

Specification
The WebSocket Protocol
Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing
Hypertext Transfer Protocol Version 2 (HTTP/2)

See also

- WebSocket API
- Evolution of HTTP
- Glossary terms:
  - HTTP
  - HTTP/2
  - QUIC